The Importance of Processor Trace in Complex Real-Time Heterogeneous Systems

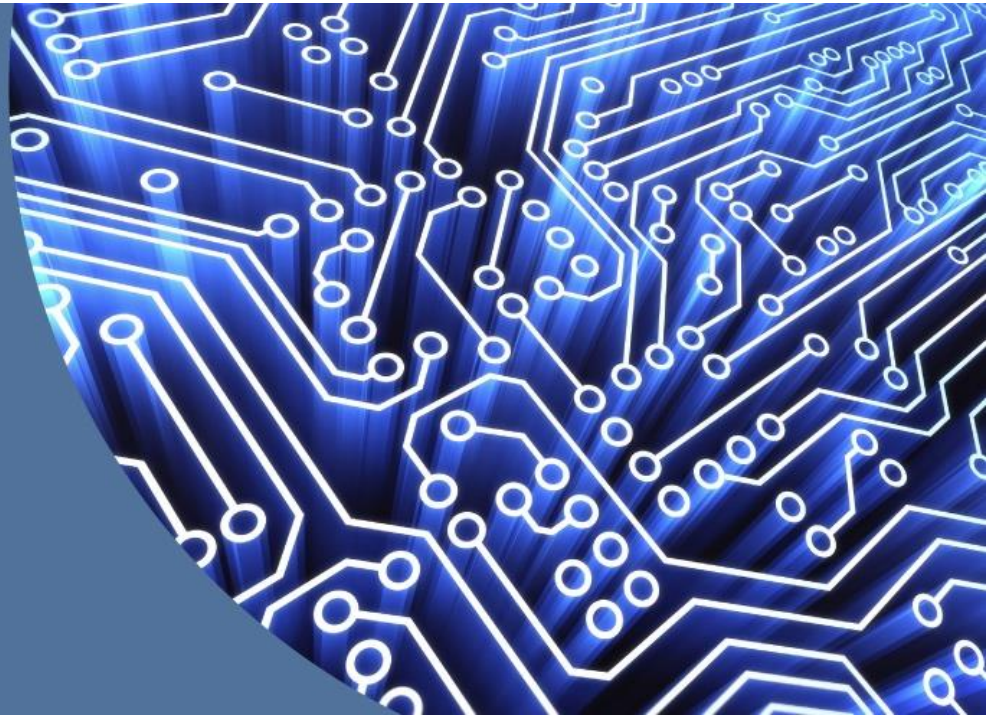
Hanan Moller, Systems Architect, UltraSoC

hanan.moller@ultrasoc.com

China RISC-V Forum

13 November 2019 – Shenzen, China

- Overview
- Processor Trace
- Algorithm
- Holistic System
- Demo System
- Summary



UltraSoC overview

- Embedded analytics
  - On-chip hardware monitors delivered as silicon intellectual property (SIP)
  - Supporting debug in-lab, & safety and security in-life
- Silicon-proven with multiple customers
- Founded 2009: VC-funded
- 35 employees; 40+ patents; HQ Cambridge UK

UltraSoC: partners and customers

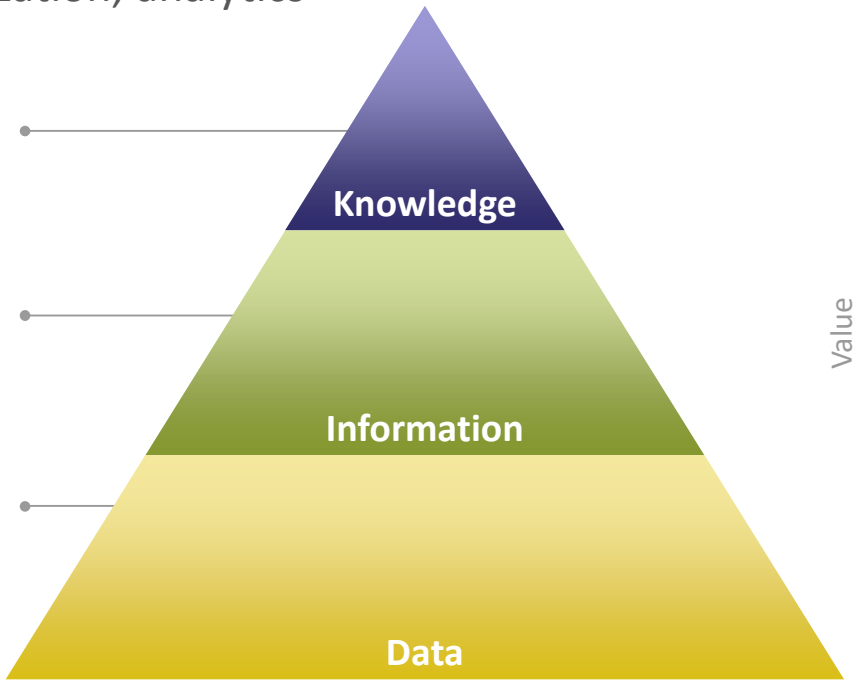
Actionable insights across the whole SoC

Debug, optimization, analytics

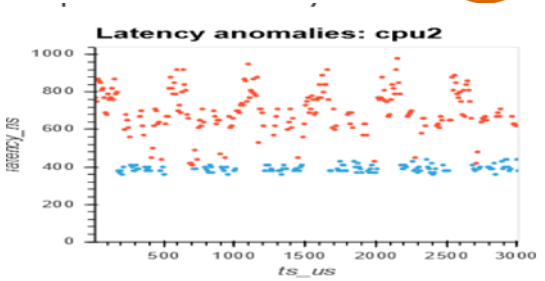
UltraSoC delivers actionable *insights*

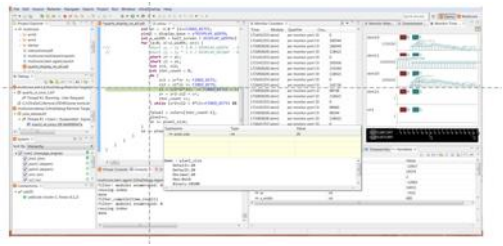
With system-wide *understanding*

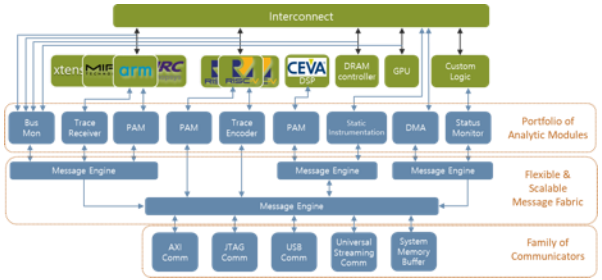
From rich *data* across the whole SoC



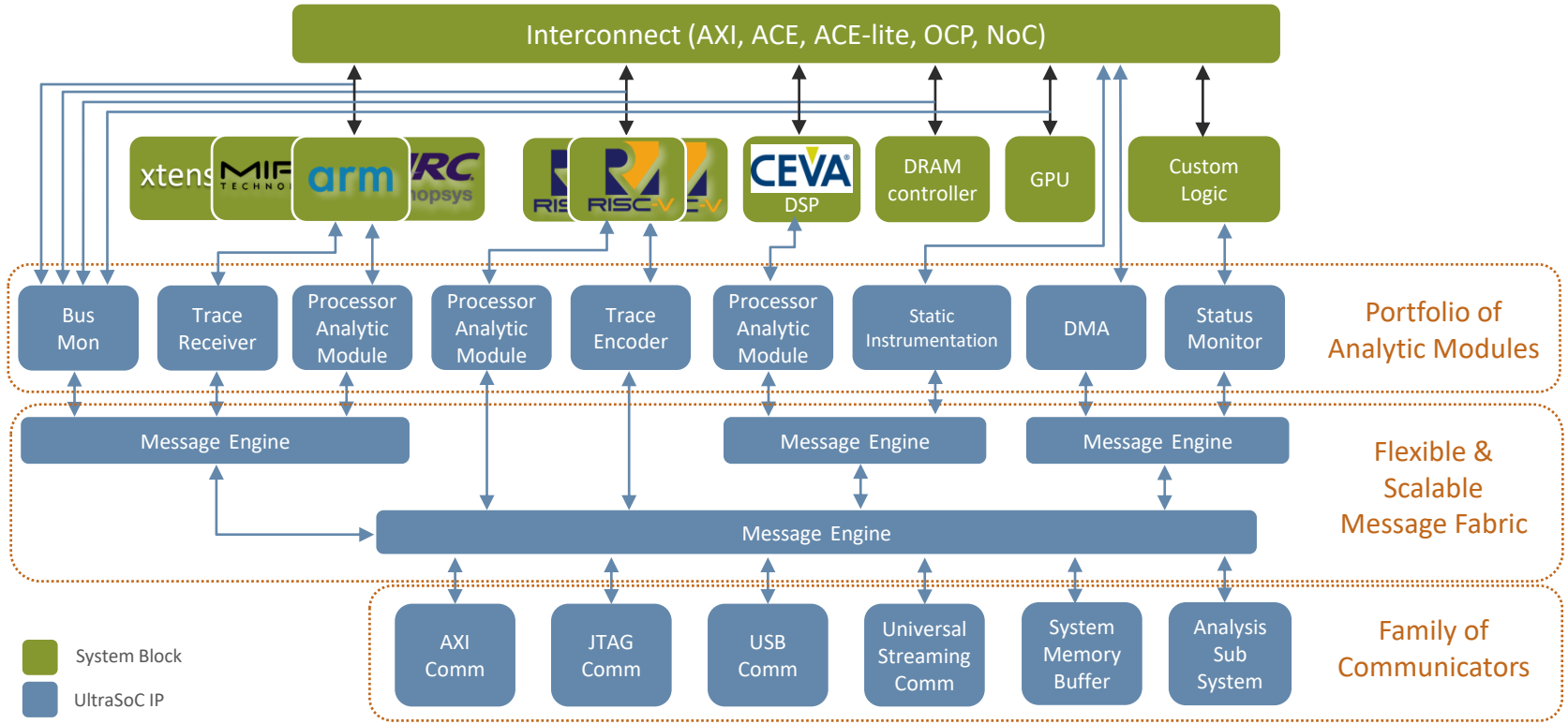
UltraSoC enables *full* visibility of SoC





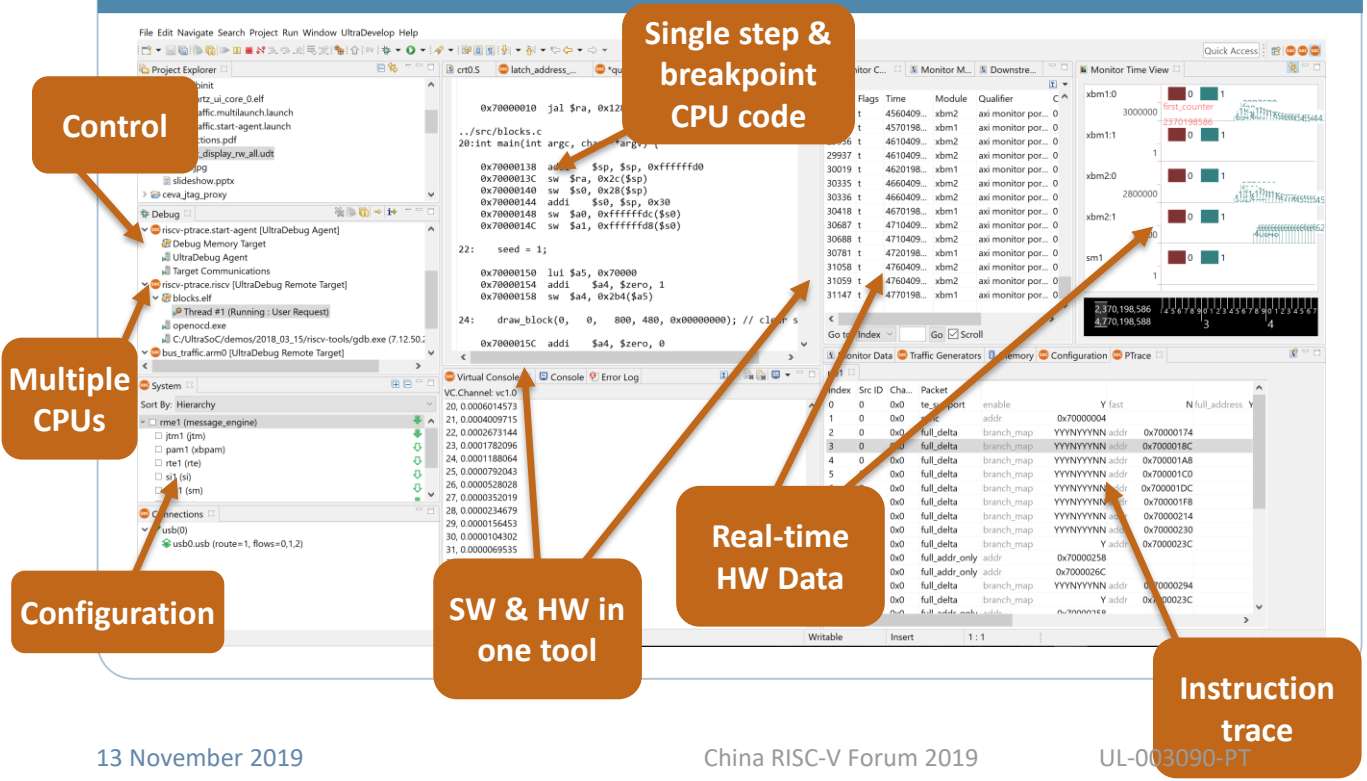


Advanced monitoring/debug for the whole SoC



Software tools for data-driven insights

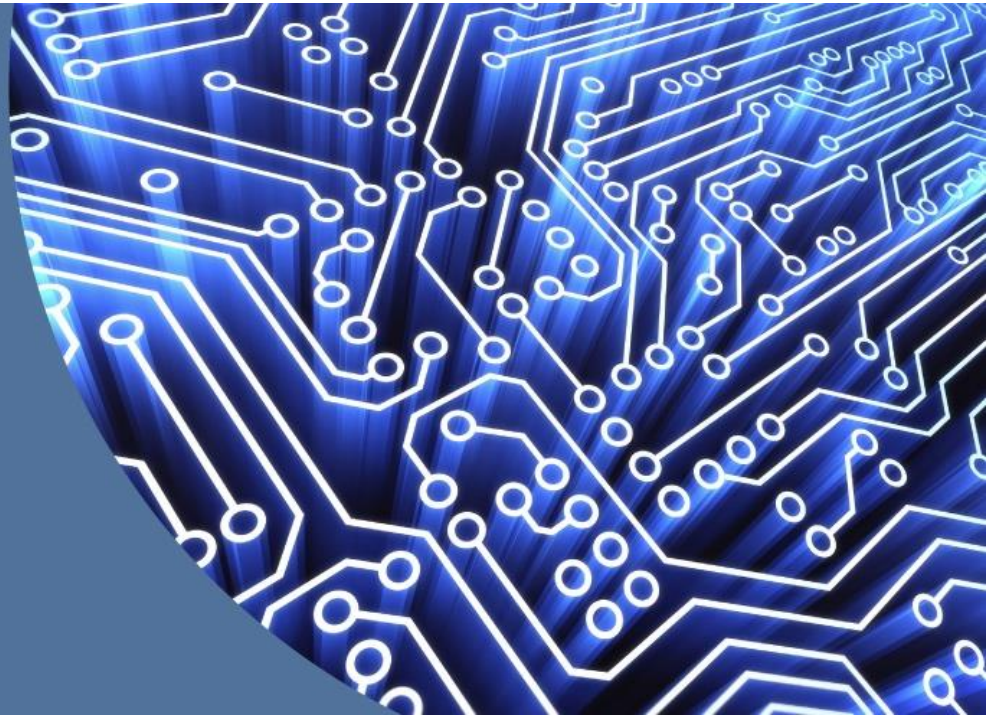
Eclipse based UltraDevelop 2 IDE



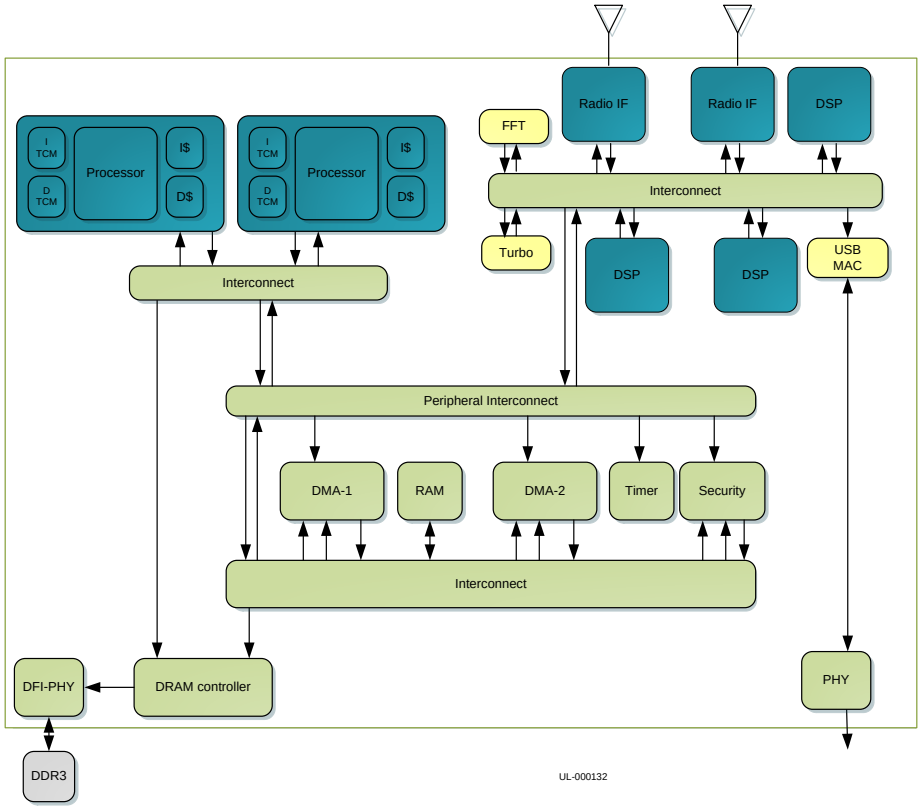
Third Party Tool Vendor Partnerships

arm
cadence
CEVA
Green Hills SOFTWARE
eclipse
IMG Systems
imperas
LAUTERBACH DEVELOPMENT TOOLS
Mentor A Siemens Business
percepio
SYNOPSYS
TELEDYNE LECROY Everywhereyoulook

Processor Branch Trace

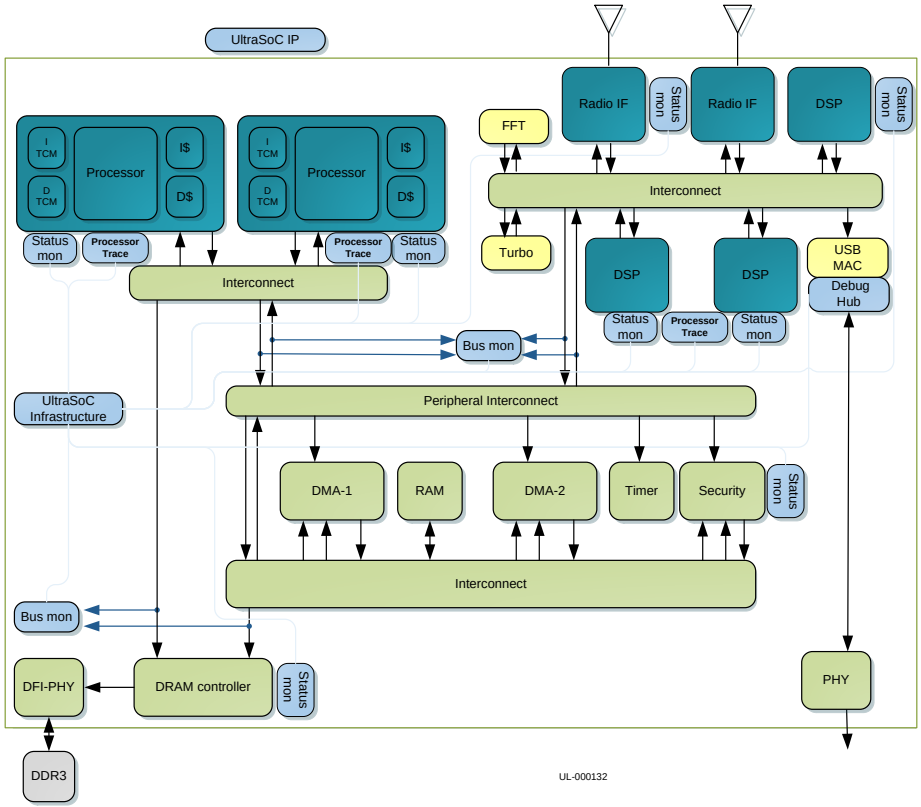


Complex Systems are … not trivial to debug



- Software sometimes does not behave as expected
  - Many contributing factors, e.g., interactions with other cores' software, peripherals, real-time events, poor implementation or some combination of all of the above
- Using a debugger is not always possible
  - Real-time behaviour is affected
- Providing **non-intrusive** visibility of program execution is important
  - This needs to be done without swamping the system with vast amounts of data

Complex Systems … need non-intrusive visibility



- Non-intrusive visibility can be achieved with Processor Trace
- Other non-intrusive methods and tools for visibility include
  - Monitoring bus transactions
  - Observing internal signals
- A system-wide debug infrastructure is required, including:
  - Flexible cross-trigger mechanisms
  - Off-chip communication paths

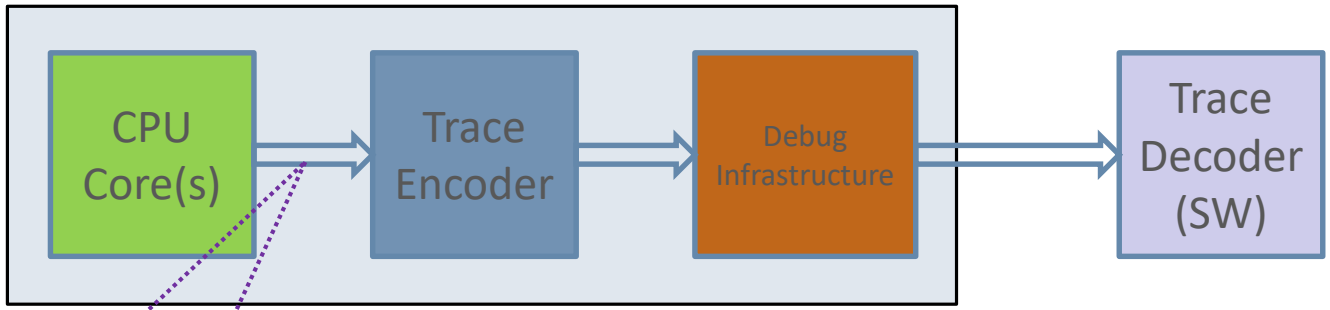
Processor Branch Trace

- Trace execution progress from a known start address by communicating deltas taken by the program.
- Deltas are typically due to instructions such as branch, jump, call and return; interrupts and exceptions are also types of deltas.
- The instructions between the deltas are assumed to be executed sequentially
  - True for ISAs, such as RISC-V, where all instructions are executed unconditionally
  - True when instruction execution can be determined based on the program (e.g., not predicated)
- The trace needs to report only:
  - Whether or not a branch was taken – the destination address known from execution binary
  - The destination address of taken indirect branches or jumps – the destination address not known statically

Interrupts and Exceptions

- Interrupts generally occur asynchronously to the program's execution rather than intentionally as a result of a specific instruction or event.
- Exceptions can be thought of in the same way, even though they can be typically linked back to a specific instruction address.
- The decoder generally does not know where an interrupt occurs in the instruction sequence, so the trace encoder must report the address where normal program flow ceased, as well as give an indication of the asynchronous destination which may be as simple as reporting the exception type.
- When an interrupt or exception occurs, or the processor is halted, the final instruction executed beforehand must be traced.

Trace Encoder Ingress Interface



Signal	Function
iretire	Instruction has retired
itype [ITYPELEN-1:0]	Termination type of instruction block
cause [CAUSELEN-1:0]	Exception cause
tval [XLEN-1:0]	Exception data
priv [PRIVLEN-1:0]	Privilege mode during execution
iaddr [XLEN-1:0]	The address of the instruction

This shows the mandatory interface signals.

Optional information can include the instruction context.

The interface supports side-band signals, for example, to indicate core is in reset, halted, or stalled.

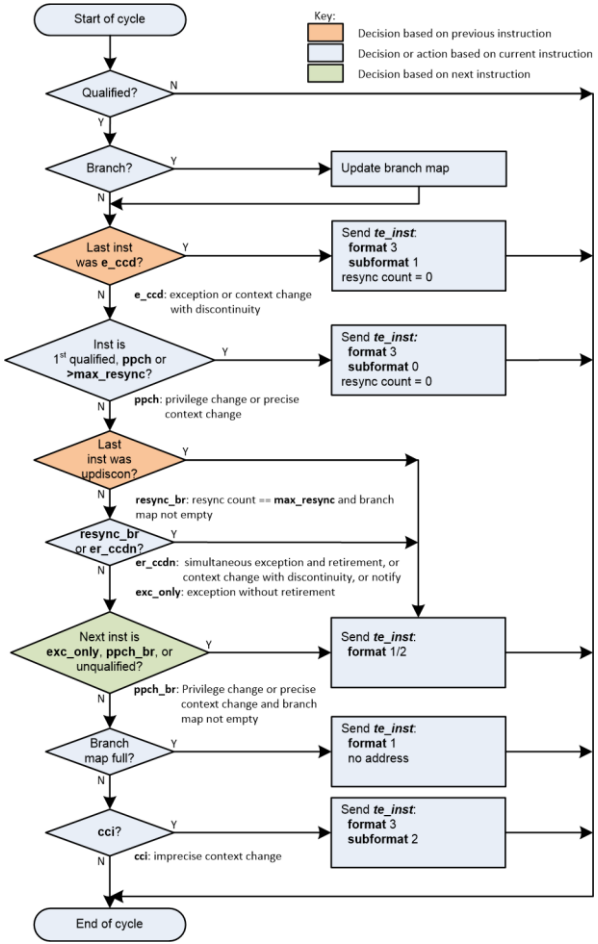
Trace Encoder Output

The Trace Encoder sends a packet containing one of the following:

1. Update – a branch map with optional differential address
2. Update – a branch map with optional full address
3. Update – address only (differential)
4. Synchronise - a context with or without a full current address

The above ensures an efficient packing to reduce data being routed on and subsequently transported off-chip

Instruction Trace Algorithm



- Formats 0 and 1 : branch map and address
  - Differential or full address
- Format 2 : address only
- Format 3 : sync packet
  - Subformat 0 : for when starting, resynchronizing or resuming from halt.
    No ***ecause***, ***interrupt*** and ***tval***
  - Subformat 1 : for exception.
    All fields present
  - Subformat 2 : for context change.
    No ***address***, ***ecause***, ***interrupt*** and ***tval***.
- See spec in:
  https://github.com/riscv/riscv-trace-spec

Trace Control

- Controlling when trace is generated is important.
  - Helps reducing volume of trace data
- Filters are used to enable:
  - Trace within an address range
  - Start trace at an address, end trace at an address
  - Trace particular privilege level
  - Trace interrupt service routines
  - Trace for fixed period of time
  - Start or stop trace when events, external to the encoder, are detected

Thank you

Hanan Moller
hanan.moller@ultrasoc.com
www.ultrasoc.com
@UltraSoC